1. **Explain about Rest Web Services?**

   REST stands for Representational State Transfer. REST is an architectural style not a protocol. In REST Architecture everything is a resource. Restful web services are light weight, highly scalable and maintainable and are very commonly used to create APIs for web-based applications.

   Restful web services use web protocol i.e. HTTP protocol method.

   In Rest everything is a resource and it can be accessed by using Uniform Resource Identifiers (URIs).

2. **Difference between Soap and Rest Web services?**

Soap	**Rest**
SOAP stands for Simple Object Access Protocol.	REST stands for Representational State Transfer
SOAP is a protocol.	REST is an architectural style.
SOAP can't use REST because it is a protocol.	REST can use SOAP web services because it is a concept and can use any protocol like HTTP, SOAP.
SOAP uses services interfaces to expose the business logic.	REST uses URI to expose business logic.
JAX-WS is the java API for SOAP web services.	JAX-RS is the java API for Restful web services.
SOAP defines standards to be strictly followed.	REST does not define too much standards like SOAP.
SOAP permits XML data format only.	REST permits different data format such as Plain text, HTML, XML, JSON etc.
SOAP requires more bandwidth for its usage. Since SOAP Messages contain a lot of information inside of it, the amount of data transfer using SOAP is generally a lot.	REST does not need much bandwidth when requests are sent to the server. REST messages mostly just consist of JSON messages. Below is an example of a JSON message passed to a web server. You can see that the size of the message is comparatively smaller to SOAP.

3. **Difference between JAX-WS and JAX-RS?**

   JAX-WS (Java API for XML based web services): A standard way to develop a Web- Services in SOAP notation (Simple Object Access Protocol).

   JAX-RS: Java API for Restful Web Services: A standard way to develop a Web- Services in architecture style like Rest.

4. **Explain the advantages of Rest Services?**

- ✓ **Fast:** RESTful Web Services are fast because there is no strict specification like SOAP. It consumes less bandwidth and resource.
- ✓ **Language and Platform independent:** RESTful web services can be written in any programming language and executed in any platform.
- ✓ **Can use SOAP:** RESTful web services can use SOAP web services as the implementation.
- ✓ **Permits different data format:** RESTful web service permits different data format such as Plain Text, HTML, XML and JSON

5. **Explain the different implementations of Rest Services?**

   Jersey
   Rest Easy

6. **Difference between Jersey and Rest Easy?**

   Both Jersey and REST Easy provide their own implementation. The difference is that Jersey additionally provides something called Chunked Output. It allows the server to send back to the client a response in parts (chunks). This approach is very useful for long processing responses for which parts of data are available before the whole response data is prepared. In result, chunks can be pushed one by one to the client during request processing.

   Rest easy is best when we are using the JBOSS server. REST Easy is available in the JBoss server family by default.

7. **Explain few annotation you have used in Rest Service?**

   @GetMapping: This annotation is used for mapping HTTP GET requests onto specific handler methods. @GetMapping is a composed annotation that acts as a shortcut for @RequestMapping(method = RequestMethod.GET)

   @PostMapping: This annotation is used for mapping HTTP POST requests onto specific handler methods. @PostMapping is a composed annotation that acts as a shortcut for @RequestMapping(method = RequestMethod.POST)

   @PutMapping: This annotation is used for mapping HTTP PUT requests onto specific handler methods. @PutMapping is a composed annotation that acts as a shortcut for @RequestMapping(method = RequestMethod.PUT)

   @PatchMapping: This annotation is used for mapping HTTP PATCH requests onto specific handler methods. @PatchMapping is a composed annotation that acts as a shortcut for @RequestMapping(method = RequestMethod.PATCH)

@DeleteMapping: This annotation is used for mapping HTTP DELETE requests onto specific handler methods. @DeleteMapping is a composed annotation that acts as a shortcut for @RequestMapping(method = RequestMethod.DELETE).

@PathVariable: It is used to extract the values from the URI. It is most suitable for the RESTful web service, where the URL contains a path variable. We can define multiple @PathVariable in a method.

@RequestParam: It is used to extract the query parameters form the URL. It is also known as a query parameter. It is most suitable for web applications. It can specify default values if the query parameter is not present in the URL.

@RequestHeader: It is used to get the details about the HTTP request headers. We use this annotation as a method parameter. The optional elements of the annotation are name, required, value, defaultValue. For each detail in the header, we should specify separate annotations. We can use it multiple time in a method

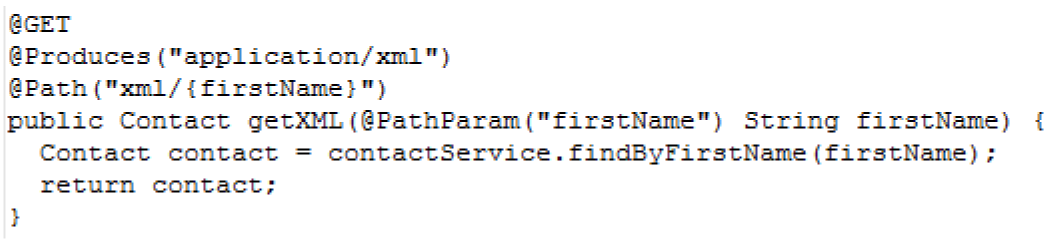
@RequestAttribute: It binds a method parameter to request attribute. It provides convenient access to the request attributes from a controller method. With the help of @RequestAttribute annotation, we can access objects that are populated on the server-side.

@RestController: This annotation is used at the class level. The @RestController annotation marks the class as a controller where every method returns a domain object instead of a view. By annotating a class with this annotation you no longer need to add @ResponseBody to all the RequestMapping method. It means that you no more use view-resolvers or send html in response. You just send the domain object as HTTP response in the format that is understood by the consumers like JSON.

@RestController is a convenience annotation which combines @Controller and @ResponseBody.

@Produces: This annotation specifies the type of output this method (or web service) will produce.

@PathParam: We can bind REST-style URL parameters to method arguments



```
@GET
@Produces("application/xml")
@Path("xml/{firstName}")
public Contact getXML(@PathParam("firstName") String firstName) {
  Contact contact = contactService.findByFirstName(firstName);
  return contact;
}
```

@QueryParam: Request parameters in query string can be accessed using @QueryParam annotation

```
@GET
@Produces("application/json")
@Path("json/companyList")
public CompanyList getJSON(@QueryParam("start") int start, @QueryParam("limit") int limit) {
  CompanyList list = new CompanyList(companyService.listCompanies(start, limit));
  return list;
}
```

@Consumes: The @Consumes annotation is used to specify the MIME media types a REST resource can consume.

```
@PUT
@Consumes("application/json")
@Produces("application/json")
@Path("{contactId}")
public RestResponse<Contact> update(Contact contact) {
...
}
```

@FormParam: The REST resources will usually consume XML/JSON for the complete Entity Bean. Sometimes, you may want to read parameters sent in POST requests directly and you can do that using @FormParam annotation. GET Request query parameters can be accessed using @QueryParam annotation.

@PUT: Annotate PUT request methods with @PUT. This is used to update the existing resource.

```
@PUT
@Consumes("application/json")
@Produces("application/json")
@Path("{contactId}")
public RestResponse<Contact> update(Contact contact) {
...
}
```

@OPTIONS: In REST OPTIONS is a method level annotation, this annotation indicates that the following method will respond to the HTTP OPTIONS request only. It is used to request, for information about the communication option available for a resource.

```
@OPTIONS
  @Produces(MediaType.APPLICATION_JSON)
  @Path("/")
  public Response optionsForBookResource() {
      return Response.status(200)
        .header("Allow","POST, PUT, GET")
        .header("Content-Type", MediaType.APPLICATION_JSON)
        .header("Content-Length", "0")
        .build();
  }
```

This method allows the client of the REST API to determine, which HTTP method (GET, HEAD, POST, PUT, DELETE) can be used for a resource identified by requested URI, without initiating a resource request by using any particular HTTP method. Response to this method are not cacheable.

8. **Difference between @PathVariable and @PathParam?**

   @PathVariable and @PathParam both are used for accessing parameters from URI Template.

   - ✓ @PathVariable is from spring and @PathParam is from JAX-RS.
   - ✓ @PathParam can use with REST only, where @PathVariable used in Spring so it works in MVC and REST.

9. **Difference between @POST, @OPTIONS, @PUT?**

   @POST: It is used to insert the new resource.
   @PUT: It is used to update the existing resource.
   @OPTIONS: It is used to request, for information about the communication option available for a resource.

10. **What is use of HATEOAS in Rest Services?**

    HATEOAS acronyms for Hypermedia as the Engine of Application State. The term hypermedia refers to content that contains a link to other forms of media like images, movies, and text.sing HATEOAS, a client interacts with a network application, whose application server provides information dynamically through Hypermedia.

    In the Spring HATEOAS project, we do not require Servlet Context and concatenate the path variable to the base URI. Instead of this, Spring HATEOAS offers three abstractions for creating the URI: ContrrollerLinkBuilder, Link, and Resource Support. We can use these abstractions to create metadata, which associates with the resource representation.

    Suppose, we have requested a GET request for localhost:8080/users/1 , it returns the details of user id 1. Along with this, it also returns a field called link that contains a link (localhost:8080/users) of all users so that consumers can retrieve all the users. This concept is called HATEOAS.

```
{
    "person": {
        "id": 2,
        "firstName": "test",
        "secondName": "one",
        "dateOfBirth": "01/01/0001 01:10",
        "profession": "im a test",
        "salary": 0
    },
    "_links": {
        "people": {
            "href": "http://localhost:8090/people"
        },
        "memberships": {
            "href": "http://localhost:8090/people/14/memberships"
        },
        "self": {
            "href": "http://localhost:8090/people/14"
        }
    }
}
```

11. How to achieve Internationalization of RESTful Services?

Internationalization is the process of designing web applications or services in such a way that it can provide support for various countries, various languages automatically without making the changes in the application. It is also known as I18N because the word internationalization has total 18 characters starting from I to N.

```
@Bean
public  LocaleResolver localeResolver()
{
SessionLocaleResolver localeResolver = new SessionLocaleResolver();
localeResolver.setDefaultLocale(Locale.US);
return localeResolver;
}
```

12. Explain Content Negotiation in Rest Services?

A resource can have many representations, mostly because there may be multiple clients expecting different representations (JSON/TEXT/XML)

Content negotiation is the process of selecting the best representation for a given response when there are multiple representations available. It is a part of HTTP that makes it possible to serve different versions of a document at the same URI.

In web API, content negotiation is performed at the server side to determine the media type formatted to be used based on return the response for an incoming request from the client-side. Content negotiation is centered on the media type and media type formatter.

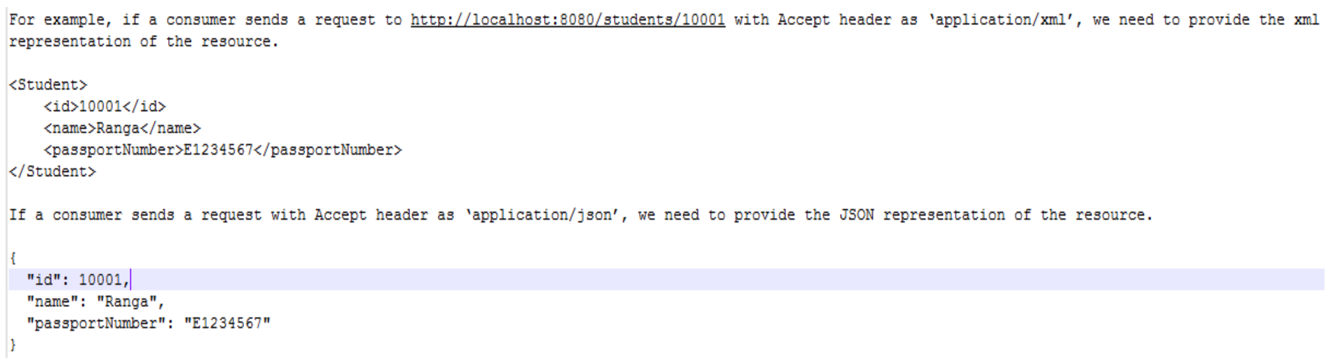
13. **Explain the different ways of Content Negotiation?**

Content negotiation using HTTP headers:

An incoming request may have an entity attached to it. To determine the type of entity server uses the HTTP request header Content-Type. There are some common content types are: application/json, application/xml, text/html, images/jpg, etc.

When a consumer sends a request, it can specify two HTTP Headers related to Content Negotiation

Content-Type: indicates the content type of the body of the request.
Accept: indicates the expected content type of the response.



```
For example, if a consumer sends a request to http://localhost:8080/students/10001 with Accept header as 'application/xml', we need to provide the xml
representation of the resource.

<Student>
    <id>10001</id>
    <name>Ranga</name>
    <passportNumber>E1234567</passportNumber>
</Student>

If a consumer sends a request with Accept header as 'application/json', we need to provide the JSON representation of the resource.

{
  "id": 10001,
  "name": "Ranga",
  "passportNumber": "E1234567"
}
```

Content negotiation using URL patterns:

There is another way to pass content type information to the server. The client can use a specific extension in resource URIs. For example, a client can request for the following:

http://demo.com/product/mobile/samsung/galaxy-s8/functions.xml
http://demo.com/product/mobile/samsung/galaxy-s8/functions.json

14. **Explain about Richardson Maturity Model?**

The Richardson maturity model is a way to grade your API according to the constraints of REST. It breaks down the principal element of the REST approach into four levels (0 to 3).

Level 0: The Swamp of POX (Plain Old XML):

To get and post the data, we send a request to the same URI, and only the POST method may be used. These APIs use only one URI and one HTTP method called POST. In short, it exposes SOAP web services in the REST style.

For example, there can be many customers for a particular company. For all the different customers, we have only one endpoint. To do any of the operations like get, delete, update, we use the same POST method.

To get the data: http://localhost:8080/customer

To post the data: http://localhost:8080/customer

Single URI + Single HTTP methods

Level 1: Resources:

It uses multiple URIs. Where every URI is the entry point to a specific resource. It exposes resources with proper URI. Level 1 tackles complexity by breaking down huge service endpoints into multiple different endpoints. It also uses only one HTTP method POST for retrieving and creating data.

For example, if we want a list of specific products, we go through the URI http://localhost:8080/products . If we want a specific product, we go through the URI http://localhost:8080/products/mobile

Many URI + Single HTTP methods

Level 2: HTTP Verbs:

At level 2, correct HTTP verbs are used with each request. It suggests that in order to be truly RESTful, HTTP verbs must be used in API. For each of those requests, the correct HTTP response code is provided.

We don't use a single POST method for all requests. We use the GET method when we request a resource, and use the DELETE method when we want to delete a resource. Also, use the response codes of the application protocol.

Many URI + Many HTTP methods

Level 3: Hypermedia Controls

Level 3 is the highest level. It is the combination of level 2 and HATEOAS. It also provides support for HATEOAS. It is helpful in self-documentation.

Many URI + Many HTTP methods + HATEOAS.

15. Explain few Rest services HTTP Status codes and their meanings?

HTTP defines these standard status codes that can be used to convey the results of a client’s request.

1xx: Informational – Communicates transfer protocol-level information.
2xx: Success – Indicates that the client’s request was accepted successfully.
3xx: Redirection – Indicates that the client must take some additional action in order to complete their request.
4xx: Client Error – This category of error status codes points the finger at clients.
5xx: Server Error – The server takes responsibility for these error status codes.

16. **Explain about RestTemplate and its methods?**

Rest Template used consume RESTful Web Services.

Get Methods:

getForObject(url, classType) – retrieve a representation by doing a GET on the URL. The response (if any) is unmarshalled to given class type and returned.
getForEntity(url, responseType) – retrieve a representation as ResponseEntity by doing a GET on the URL.
exchange(requestEntity, responseType) – execute the specified RequestEntity and return the response as ResponseEntity.

Post Methods:

postForObject(url, request, classType) – POSTs the given object to the URL, and returns the representation found in the response as given class type.
postForEntity(url, request, responseType) – POSTs the given object to the URL, and returns the response as ResponseEntity.
postForLocation(url, request, responseType) – POSTs the given object to the URL, and returns returns the value of the Location header.
exchange(url, requestEntity, responseType) – execute the specified RequestEntity and return the response as ResponseEntity.

Delete methods:

delete(url) – deletes the resource at the specified URL.

17. **Difference between Ajax and Rest?**

AJAX basically refers to making asynchronous request in JavaScript, traditionally sending/receiving XML (although nowadays, JSON is often used instead of XML). So that's the technique you use on client-side.

REST is a concept for HTTP request exchange, so you're making RESTful request calls (e.g. 'get') against the REST-API you implemented on server side.

AJAX is a set of (typically) client-sided web development techniques, while REST is an architecture style for sending and handling HTTP requests. So you can use AJAX to send RESTful requests. A REST API is typically not implemented using AJAX, but can be accessed by an AJAX client.

18. Explain about WebClient?

- ✓ Spring Framework 5 introduced a new HTTP client called WebClient.
- ✓ WebClient is a modern, alternative HTTP client to RestTemplate. Not only does it provide a traditional synchronous API, but it also supports an efficient nonblocking and asynchronous approach.
- ✓ That said, if we're developing new applications or migrating an old one, it's a good idea to use WebClient. Moving forward, RestTemplate will be deprecated in future versions.

19. What is the use of HttpHeaders in Rest Service?

HTTP headers let the client and the server pass additional information with an HTTP request or response. An HTTP header consists of its case-insensitive name followed by a colon (:), then by its value. Whitespace before the value is ignored. HttpHeaders to set the Request Headers.

```
@RequestMapping(value = "/template/products", method = RequestMethod.POST)
public String createProducts(@RequestBody Product product) {
   HttpHeaders headers = new HttpHeaders();
   headers.setAccept(Arrays.asList(MediaType.APPLICATION_JSON));
```

20. What is the use of HttpEntity in Rest Service?

Use the HttpEntity to wrap the request object. Here, we wrap the Product object to send it to the request body.

```
@RequestMapping(value = "/template/products", method = RequestMethod.POST)
public String createProducts(@RequestBody Product product) {
   HttpHeaders headers = new HttpHeaders();
   headers.setAccept(Arrays.asList(MediaType.APPLICATION_JSON));
   HttpEntity<Product> entity = new HttpEntity<Product>(product,headers);
```

21. Explain the architectural style for creating web API?

The architectural style for creating web api are

- ✓ HTTP for client server communication
- ✓ XML/JSON as formatting language
- ✓ Simple URI as the address for the services
- ✓ Stateless communication

22. What tools are used to test web services?

- ✓ SoapUI tool for testing SOAP and RESTful web services
- ✓ Postman extension for Chrome

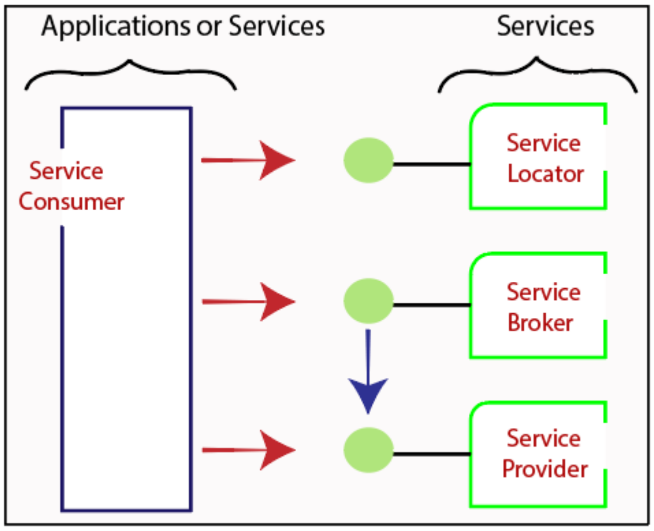
23. Explain the characteristics of REST?

- ✓ REST is stateless, therefore the SERVER has no state (or session data)
- ✓ With a well-applied REST API, the server could be restarted between two calls as every data is passed to the server
- ✓ Web service mostly uses POST method to make operations, whereas REST uses GET to access resources

24. Explain the characteristics of SOA?

- ✓ They are loosely coupled.
- ✓ They support interoperability.
- ✓ They are location-transparent
- ✓ They are self-contained

25. Explain the architecture of SOA?



Services - The services are the logical entities defined by one or more published interfaces.

Service provider - It is a software entity that implements a service specification.
Service consumer - It can be called as a requestor or client that calls a service provider. A service consumer can be another service or an end-user application.
Service locator - It is a service provider that acts as a registry. It is responsible for examining service provider interfaces and service locations.
Service broker - It is a service provider that pass service requests to one or more additional service providers.

26. Explain web service architecture?

The web service framework includes three different layers.

The roles of these layers are:

Service Provider: Role of Service provider is to make the web service which makes it accessible to the client applications over the Web.
Service Requestor: Service requestor refers to any consumer of web service like any client application. Client applications are written in any language contact web service for any functionality by sending XML request over the available network connection.
Service Registry: Service Registry is the centralized directory System which helps to locate the web services for client applications. Used to find the existing web services, as well as developers, can also create the brand new one web service also.